Контрольная работа

**Вариант № 5**

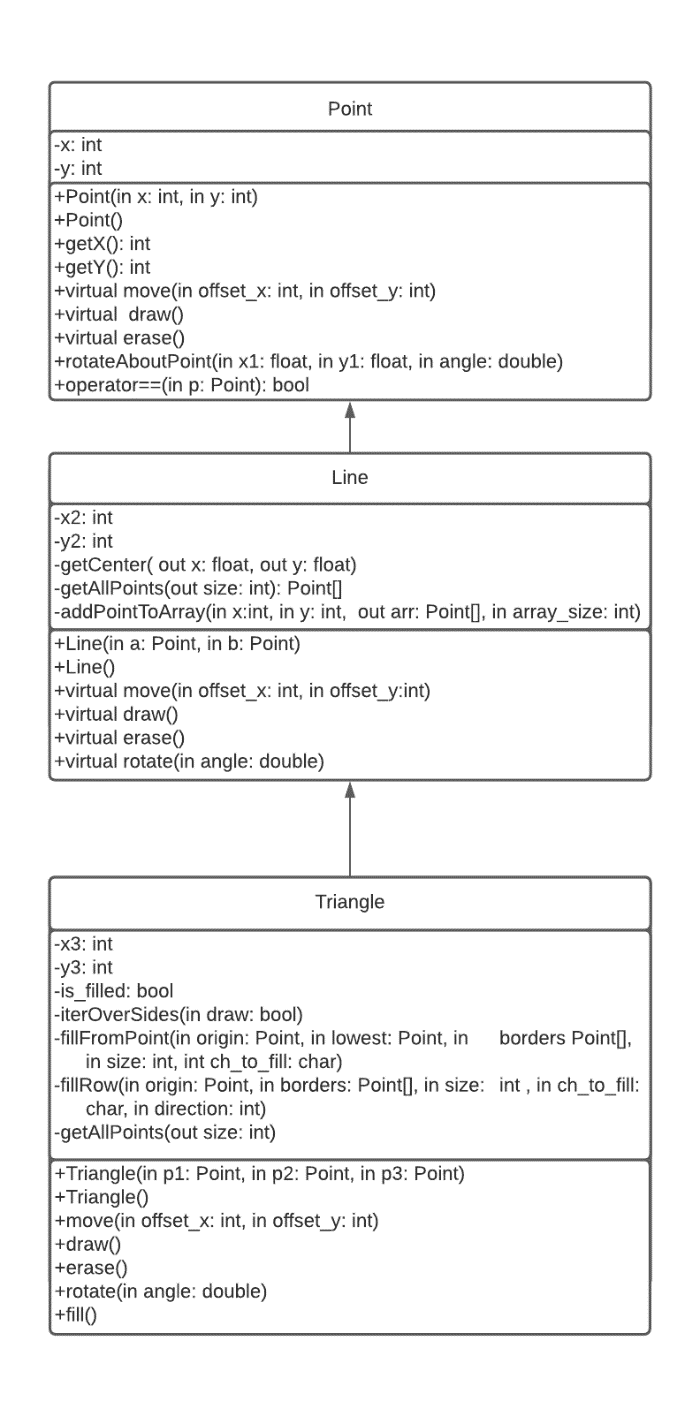
**Цель работы**

Изучение механизма наследования классов. Получение навыков в написании программ с использованием механизма наследования. Изучение и получение навыков в использовании виртуальных функций, а также шаблонов классов и функций.

**Задания**

1. Точка : отрезок : треугольник (нарисовать, стереть, закрасить, передвинуть, повернуть). Определить классы и их иерархию, а также продемонстрировать использование введенных конструкций при работе.

**Структурная схема классов**



#### Исходный текст программы

class Point{ friend class Line;

friend class Triangle;

protected: int x,y;//координаты на экране

public: Point(int x, int y);

Point(); //конструктор по умолчанию

int getX(); int getY();

virtual void move(int offset\_x, int offset\_y); //переместить

virtual void draw(); //нарисовать

virtual void erase(); //стереть

//вращение точки относительно другой

void rotateAboutPoint(float x1, float y1, double angle);

bool operator==(Point p); };

class Line : public Point { friend class Triangle;

protected: int x2, y2;//координаты конца отрезка (последняя точка не отрисовывается)

//координаты начала отрезка - поля базового класса Point (x,y)

void getCenter(float \*x, float \*y);

private: Point \*getAllPoints(unsigned int \*size); //генерирует массив точек

//добавляет точку в массив

void addPointToArray(int x, int y, Point \*\*arr, unsigned int array\_size);

public: Line(Point a, Point b);

Line();

virtual void move(int offset\_x, int offset\_y) override;

virtual void draw() override;

virtual void erase() override;

virtual void rotate(double angle); };

class Triangle : public Line { private:

int x3, y3; //координаты третьей вершины треугольника

//две остальные вершины - поля x,y базового класса Point и поле x2, y2 базового класса Line

bool is\_filled;//закрашен

//отрисовка(если draw = true) и стирание сторон треугольинка

void iterOverSides(bool draw);

//начинает закраску треугольинка с точки(нужно для заливки) пока не дойдёт до высоты lowest

void fillFromPoint(Point origin, Point lowest, Point \*borders, int size, char ch\_to\_fill);

Point\* getAllPoints(int \*size);

public: Triangle(Point p1, Point p2, Point p3);

Triangle();

void move(int offset\_x, int offset\_y) override;

void draw() override;

void erase() override;

void rotate(double angle) override;

void fill(); //закрасить };

**// реализация**

**#include "geometrics.h"**

**#include <windows.h>**

**#include <iostream>**

**#include <math.h>**

**#include <algorithm>**

**void moveConsoleCursor(int x, int y);**

**//возвращает отрезок, принадлежащий треугольинку на данной высоте**

**//если такого нет, то NULL**

**Line\* getLineWithEqualHeight(int y, Point \*points, int points\_num);**

**//class Point**

**Point::Point(int x, int y) { this->x = x; this->y = y; }**

**Point::Point() : Point(0, 0) { }**

**int Point::getX() { return x; }**

**int Point::getY() { return y; }**

**void Point::move(int offset\_x, int offset\_y) { this->x += offset\_x; this->y += offset\_y; }**

**void Point::draw() { moveConsoleCursor(x, y); std::cout << '\*'; }**

**void Point::erase() { moveConsoleCursor(x, y); std::cout << ' '; }**

**void Point::rotateAboutPoint(float x1, float y1, double angle) {**

**x = (int)((x - x1) \* cos(angle) -(y - y1) \* sin(angle) + x1);**

**y = (int)((x - x1) \* sin(angle) + (y - y1) \* cos(angle) + y1); }**

**bool Point::operator==(Point p) { return x == p.x && y == p.y; }**

**//class Line**

**Line::Line(Point a, Point b) : Point(a), x2(b.x), y2(b.y) { }**

**Line::Line() : Line(Point(), Point()) { }**

**void Line::move(int offset\_x, int offset\_y) {**

**Point::move(offset\_x, offset\_y); //передвигаем начало отрезка**

**x2 += offset\_x; y2 += offset\_y; //**и конец **}**

**void Line::draw() {**

**unsigned int size;**

**Point \*points = getAllPoints(&size);**

**for (int i = 0; i < size; i++) points[i].draw();**

**delete[] points; }**

**void Line::erase() {**

**unsigned int size;**

**Point \*points = getAllPoints(&size);**

**for (int i = 0; i < size; i++) points[i].erase();**

**if(points) delete[] points; }**

**//Находим все точки, принадлежащие отрезку по алгоритму Брезенхэма**

**Point\* Line::getAllPoints(unsigned int\* size) {**

**\*size = 0;**

**Point \*points = NULL;**

**//разница между началом и концом отрезка (по модулю)**

**int difference\_x = abs(x2 - x); int difference\_y = abs(y2 - y);**

**int error = 0.f; //ошибка нужна для того чтобы смещаться по y координате**

**int delta\_error = difference\_y + 1;**

**int current\_y = y; int delta\_y = y2 - y;**

**if (delta\_y > 0) delta\_y = 1;**

**else if (delta\_y < 0) delta\_y = -1;**

**if (difference\_x) //если отрезок не перпендикулярен горизонтальной стороне экрана**

**{**

**int delta\_x = x2 - x;**

**int sign = delta\_x > 0 ? 1 : -1; //знак разницы между начальной x-координатой и конечной**

**for (int current\_x = x;**

**x2 < current\_x || current\_x < x2;**

**current\_x += sign) {**

**addPointToArray(current\_x, current\_y, &points, \*size);**

**(\*size)++;**

**error += delta\_error;**

**while (error >= difference\_x + 1) {**

**//если ошибка слишком большая, это значит что линия практически вертикальна поэтому нужно для каждого x отрисовывать колонку точек**

**current\_y += delta\_y;**

**addPointToArray(current\_x, current\_y, &points, \*size);**

**(\*size)++; error -= difference\_x + 1;**

**}**

**}**

**}**

**else //! difference\_ч**

**if (difference\_y)**

**for (int current\_y = y;**

**y2 < current\_y || current\_y < y2;**

**current\_y += delta\_y) {**

**addPointToArray(x, current\_y, &points, \*size);**

**(\*size)++;**

**}**

**return points; }**

**void Line::addPointToArray(int x, int y, Point \*\*arr, unsigned int array\_size) {**

**Point point(x, y);**

**Point \*temp = \*arr; \*arr = new Point[array\_size + 1];**

**if(array\_size) memcpy(\*arr, temp, array\_size \* sizeof(Point));**

**\*(\*arr + array\_size) = point;**

**if(temp) delete[] temp; }**

**void Line::rotate(double angle) { float center\_x, center\_y;**

**getCenter(&center\_x, &center\_y);**

**Point::rotateAboutPoint(center\_x, center\_y, angle);**

**Point second\_p(x2, y2); //конец отрезка**

**second\_p.rotateAboutPoint(center\_x, center\_y, angle);**

**x2 = second\_p.x; y2 = second\_p.y; }**

**void Line::getCenter(float \*x, float \*y) { \*x = (float)(this->x + x2) / 2.f; \*y = (float)(this->y + y2) / 2.f; }**

**//class Triangle**

**Triangle::Triangle(Point p1, Point p2, Point p3) : Line(p1, p2), x3(p3.x),**

**y3(p3.y), is\_filled(false) { }**

**Triangle::Triangle() : Line(Point(), Point()), x3(0), y3(0) { }**

**void Triangle::move(int offset\_x, int offset\_y) {Line::move(offset\_x, offset\_y); x3 += offset\_x; y3 += offset\_y;}**

**void Triangle::draw() { iterOverSides(true); }**

**void Triangle::erase() { iterOverSides(false); }**

**void Triangle::iterOverSides(bool draw) { //создаём 2 стороны треугольника**

**Line B(Point(x2, y2), Point(x3, y3)); Line C(Point(x, y), Point(x3, y3));**

**//и отрисовываем, либо стираем if (draw) { Line::draw(); B.draw(); C.draw(); }**

**else { Line::erase(); B.erase(); C.erase(); }**

**if (is\_filled) //если треугольник закрашенный**

**{ Point highest,lowest; if (y3 <= y2 && y3 <= y) { highest = Point(x3, y3); lowest = y2 >= y ? Point(x2, y2) : Point(x, y); }**

**else if (y2 <= y && y2 <= y3) { highest = Point(x2, y2);**

**lowest = y3 >= y ? Point(x3, y3) : Point(x, y); }**

**else { highest = Point(x, y); lowest = y3 >= y2 ? Point(x3, y3) : Point(x2, y2); }**

**int size;**

**Point \*all\_points = getAllPoints(&size);**

**fillFromPoint(highest, lowest, all\_points, size, draw ? '\*' : ' ');**

**delete[] all\_points;**

**}**

**}**

**Point\* Triangle::getAllPoints(int \*size) {**

**//получаем точки всех сторон треугольинка**

**Point \*line1, \*line2, \*line3; unsigned int size1, size2, size3;**

**line1 = Line::getAllPoints(&size1); Line B(Point(x2, y2), Point(x3, y3));**

**Line C(Point(x, y), Point(x3, y3));**

**line2 = B.getAllPoints(&size2); line3 = C.getAllPoints(&size3);**

**\*size = size1 + size2 + size3;**

**Point \*all\_points = new Point[\*size];**

**memcpy(all\_points, line1, size1 \* sizeof(Point)); memcpy(all\_points + size1, line2, size2 \* sizeof(Point));memcpy(all\_points + size2, line3, size3 \* sizeof(Point));**

**delete[] line1; delete[] line2; delete[] line3;**

**return all\_points;**

**}**

**void Triangle::fillFromPoint(Point origin, Point lowest, Point \*borders, int size, char ch\_to\_fill) {**

**Line\* line;**

**while ((line = getLineWithEqualHeight(origin.y++, borders, size)) != NULL) {**

**ch\_to\_fill == '\*' ? line->draw() : line->erase(); delete line; } }**

**void Triangle::rotate(double angle) {**

**float center\_x, center\_y;**

**center\_x = (float)(x + x2 + x3) / 3.f; center\_y = (float)(y + y2 + y3) / 3.f;**

**Point::rotateAboutPoint(center\_x, center\_y, angle);**

**Point p2(x2, y2), p3(x3,y3);**

**p2.rotateAboutPoint(center\_x, center\_y, angle);**

**p3.rotateAboutPoint(center\_x, center\_y, angle);**

**x2 = p2.x; y2 = p2.y; x3 = p3.x; y3 = p3.y;**

**}**

**void Triangle::fill() { is\_filled = true; }**

**static void moveConsoleCursor(int x, int y){**

**HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**COORD p = { x, y };**

**SetConsoleCursorPosition(handle, p); }**

**Line\* getLineWithEqualHeight(int y, Point\* points, int points\_num) {**

**Line \*line = NULL;**

**Point \*p1 = NULL, \*p2 = NULL;** //начало и конец отрезка

**for (int i = 0; i < points\_num; i++) {**

**if (points[i].getY() == y) {**

**if (!p1) p1 = points + i;**

**else { p2 = points + i; break; }**

**}**

**}**

**if (p1 && p2) { line = new Line(\*p1, \*p2); }**

**return line;**

**}**

**#include "geometrics.h"**

**#include <iostream>**

**#define \_USE\_MATH\_DEFINES**

**#include <math.h>**

**#include <Windows.h>**

**int main() {**

**Triangle triangle(Point(1, 0), Point(1, 10), Point(11, 10));**

**Triangle triangle2;**

**Line line1(Point(37, 10), Point(47, 10));**

**Line line2 = line1;**

**Point point(1, 1);**

**triangle.move(5, 5);**

**triangle2 = triangle;**

**triangle.fill();**

**triangle.draw();**

**triangle2.rotate(M\_PI / 2 - 1); //поворот на 32 градуса**

**triangle2.move(15, 0);**

**triangle2.draw();**

**point.draw();**

**line1.draw();**

**line2.rotate(M\_PI / 4); // поворот на 45 градусов**

**line2.draw();**

**Sleep(10000);**

**triangle.erase();**

**triangle2.erase();**

**line1.erase();**

**line2.erase();**

**point.erase();**

**getchar();**

**return 0;**

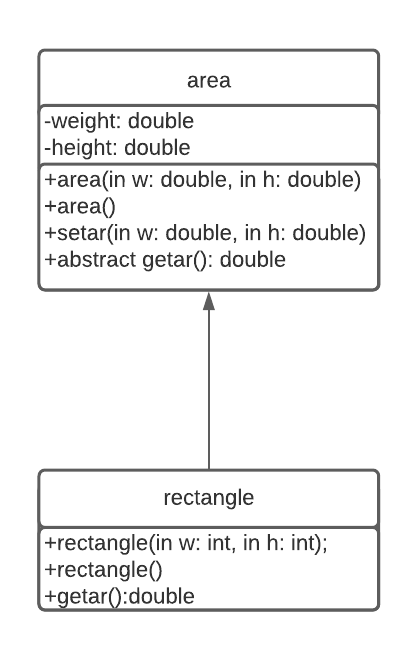
**}**

**Результат работы программы**



2. Напишите программу создания базового класса **area.** В этом классе должны храниться две переменные типа **double**, определяющие размеры фигуры (например, высота и ширина), определена функция **setar( ),** задающая значения переменным, и определена виртуальная функция **getar( ).** Создайте два производных класса **rectangle** и **triangle**, которые наследуют класс **area**. Функция **getar( )** должна быть переопределена в производных классах так, чтобы возвращать площадь фигуры, вид которой задается в производном классе.

**Структурная схема класса**



**Исходный текст программы**

**class area { protected: double weight, height;**

**public: area(double w, double h); area();**

**void setar(double w, double h);**

**virtual double getar() = 0; };**

**class rectangle : public area { public: rectangle(int w, int h);**

**rectangle(); double getar() override; };**

**class triangle : public area { public: triangle(int w, int h); triangle(); double getar() override; };**

**// реализация классов**

**#include "shapes.h"**

**//class area**

**area::area(double w, double h) : weight(w), height(h) { }**

**area::area() : area(0., 0.) { }**

**void area::setar(double w, double h){ weight = w; height = h; }**

**//class rectangle**

**rectangle::rectangle(int w, int h) : area(w, h) { }**

**rectangle::rectangle() : area() { }**

**double rectangle::getar() { return weight \* height; }**

**//class triangle**

**triangle::triangle(int w, int h) : area(w, h) { }**

**triangle::triangle() : area() { }**

**double triangle::getar() { return weight \* height / 2; }**

**#include "shapes.h"**

**#include <iostream>**

**int main() {**

**int w1, h1, w2, h2;**

**//вводим ширину и высоту прямоугольника**

**std::cin >> w1 >> h1;**

**rectangle rect;**

**rect.setar(w1, h1);**

**std::cout << "Area of rectangle = " << rect.getar() << std::endl;**

**//вводим основание треугольинка и высоту**

**std::cin >> w2 >> h2;**

**triangle t(w2, h2);**

**std::cout << "Area of triangle = " << t.getar() << std::endl;**

**return 0;**

**}**

**Результат работы программы**

1.Входные данные: 12 10.

Вывод: Area of rectangle = 120

2. Входные данные: 14 15

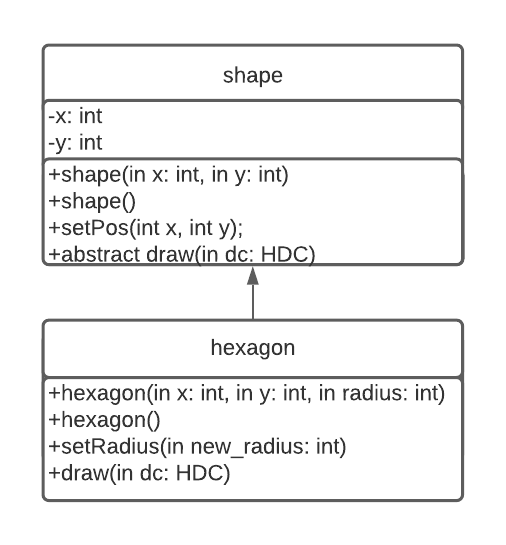
Вывод: Area of triangle = 105

3. Используя абстрактный базовый класс "фигура", разработанный самостоятельно, определить класс, описанный ниже. Предусмотреть следующие действия с объектами создаваемого класса:

1. динамическое создание объектов, в том числе создание массивов объектов;
2. динамическое уничтожение объектов;
3. изменение параметров объекта (цвет, размер, координаты и т.п.);
4. отображение объектов на экране.

Вариант задания: Правильный шестиугольник.

**Структурная схема классов**



**Исходный текст программы**

**#include <Windows.h>**

**class shape { protected: int x, y;//координаты центра фигуры**

**public: shape(int x, int y);**

**shape();**

**void setPos(int x, int y);**

**virtual void draw(HDC dc) = 0; };**

**class hexagon : public shape {**

**private: int radius;**

**public: hexagon(int x, int y, int radius);**

**hexagon();**

**void setRadius(int new\_radius);**

**void draw(HDC dc); };**

**// реализация классов**

**#include "hexagon.h"**

**#include <Windows.h>**

**#define \_USE\_MATH\_DEFINES**

**#include <math.h>**

**//class shape**

**shape::shape(int x, int y) { this->x = x; this->y = y; }**

**shape::shape() : shape(0,0) { }**

**void shape::setPos(int x, int y) { this->x = x; this->y = y; }**

**//class hexagon**

**hexagon::hexagon(int x, int y, int radius) : shape(x,y) { this->radius = radius; }**

**hexagon::hexagon() : hexagon(0,0,0) { }**

**void hexagon::setRadius(int new\_radius) { radius = new\_radius; }**

**void hexagon::draw(HDC dc) { double angle = M\_PI / 3; //угол между диагоналями**

**double half\_a = angle / 2;**

**POINT points[6];**

**int side\_height = (long)(radius \* sin(half\_a));//высота боковых углов**

**int side\_dist = (long)(radius \* cos(half\_a));//расстояние от центра до сторон**

**points[0] = { x - side\_dist, y + side\_height };**

**points[1] = { x, y + radius };**

**points[2] = { x + side\_dist, y + side\_height };**

**points[3] = { x + side\_dist, y - side\_height };**

**points[4] = { x, y - radius };**

**points[5] = { x - side\_dist, y - side\_height };**

**SelectObject(dc, GetStockObject(WHITE\_PEN));**

**Polygon(dc, points, 6); //рисуем шестиугольник**

**}**

**#include <Windows.h>**

**#include "hexagon.h"**

**#include <iostream>**

**int main() {**

**HWND hWnd = GetConsoleWindow();**

**HDC hDC = GetDC(hWnd);**

**int radius, number, distance; //радиус шестиугольников, количество и расстояние между ними**

**//вводим радиус шестиугольников и их количество**

**std::cout << "radius: "; std::cin >> radius;**

**std::cout << "\nnumber: "; std::cin >> number;**

**system("CLS");**

**distance = radius \* 2;**

**shape \*\*shapes = new shape\*[number];**

**//создаём ряд шестиугольников по введённым данным и отрисовываем**

**for (int i = 0; i < number; i++){ shapes[i] = new hexagon(distance + i \* distance, distance, radius);**

**shapes[i]->draw(hDC); }**

**//удаляём выделенную память**

**for (int i = 0; i < number; i++) delete shapes[i];**

**delete[] shapes;**

**Sleep(10000);**

**return 0; }**

**Результат работы программы**

Входные данные 30, 5

Вывод:

****

4.Создать шаблон класса «однонаправленный линейный список». Использовать его при решении следующей задачи:

Составить программу, которая содержит динамическую информацию о заявках на авиабилеты. Каждая заявка содержит:

пункт назначения; номер рейса;

фамилию и инициалы пассажира;

желаемую дату вылета.

Программа должна обеспечивать:

хранение всех заявок в виде списка;

добавление заявок в список; удаление заявок;

вывод заявок по заданному номеру рейса и дате вылета;

вывод всех заявок.

**Исходный текст программы**

**#include <iostream>**

**template<class T>**

**class list { private: list \*next; T value;**

**public: list();**

**//добавляет элемент**

**void add(T val);**

**//удаляет элемент**

**void del(T val);**

**void printAll();**

**//печатает все элементы, которые соответсвуют условию, проверяемому в функции pr**

**template<typename Predictor>**

**void printAllWhere(Predictor pr);**

**T popBack();**

**~list(); };**

**// реализация**

**//class list**

**template<class T>**

**list<T>::list() : next(NULL) { }**

**template<class T>**

**void list<T>::add(T val) {**

**if (!next) { next = new list<T>(); next->value = val; }**

**else { next->add(val); }**

**}**

**template<class T>**

**void list<T>::del(T val) {**

**if (!next) return;**

**if (next->value == val) {**

**list<T> \*temp = next;**

**next = temp->next;**

**temp->next = NULL;**

**delete temp;**

**}**

**else next->del(val);**

**}**

**template<class T>**

**void list<T>::printAll() {**

**value.print();**

**if (next) next->printAll();**

**}**

**template<class T>**

**template <typename Predictor>**

**void list<T>::printAllWhere(Predictor pr) {**

**if (pr(value)) value.print();**

**if (next) next->printAllWhere(pr);**

**}**

**template<class T>**

**T list<T>::popBack() {**

**if (!next) throw "Empty list";**

**if (next->next == NULL) {**

**T last = next->value;**

**delete next;**

**next = NULL;**

**return last;**

**}**

**return next->popBack();**

**}**

**template<class T>**

**list<T>::~list() { if(next) delete next; }**

**#include <iostream>**

**#include "list.h"**

**#include <string>**

**class bilet //авиабилеты**

**{ private: const char \*destination; //пункт назначения**

**const char \*surname; //фамилия**

**int flight\_number;**

**char name, patronymic; //имя, отчество (инициалы)**

**int year, month, day; //дата вылета**

**public: bilet(const char \*destination, const char \*surname, int flight\_number, char name,**

**char patronymic, int year, int month, int day);**

**bilet();**

**bool operator==(bilet b);**

**void print();**

**int getFlightNum();**

**int** getYear**();**

**int getMonth();**

**int getDay();**

**};**

**// реализация**

**bilet::bilet(const char \*destination, const char \*surname, int flight\_number, char name,**

**char patronymic, int year, int month, int day)**

**{**

**this->destination = destination;**

**this->surname = surname;**

**this->flight\_number = flight\_number;**

**this->name = name;**

**this->patronymic = patronymic;**

**this->year = year;**

**this->month = month;**

**this->day = day;**

**}**

**bilet::bilet() : bilet(NULL, NULL, 0, '\0', '\0', 0, 0, 0) { }**

**bool bilet::operator==(bilet b) {**

**return (strcmp(destination, b.destination)) == 0 &&**

**(strcmp(surname, b.surname)) == 0 && flight\_number == b.flight\_number &&**

**name == b.name && patronymic == b.patronymic && year == b.year &&**

**month == month && day == b.day;**

**}**

**void bilet::print() {**

**if (!surname || (!destination)) //пустой билет**

**return;**

**std::cout << surname << ' ' << name << '.' << patronymic << ".:" << std::endl**

**<< "\tDestination: " << destination << std::endl**

**<< "\tFlight nubmer: " << flight\_number << std::endl**

**<< "\tDate: " << month << '.' << day << '.' << year << std::endl << std::endl;**

**}**

**int bilet::getFlightNum() { return flight\_number; }**

**int bilet::**getYear**() { return year; }**

**int bilet::getMonth() { return month; }**

**int bilet::getDay() { return day; }**

**//предикат, в котором проверяется нужная дата в билете**

**bool isSuitableData(bilet &b) {**

**return b.getFlightNum() == 324 && b.getYear() == 2021 && b.getMonth() == 6 && b.getDay() == 24;**

**}**

**int main() {**

**list<bilet> l;**

**bilet Ivanov("Moscow", "Ivanov", 324, 'R', 'V', 2021, 6, 24),**

**Petrov("Paris", "Petrov", 324, 'B', 'N', 2021, 6, 24);**

**l.add(Ivanov);**

**l.add(Petrov);**

**l.add(bilet("Kazan", "Sidorova", 144, 'V', 'E', 2021, 7, 14));**

**l.add(bilet("Antalya", "Elmov", 574, 'I', 'B', 2021, 8, 24));**

**l.add(bilet("Vladivostok", "Chernov", 574, 'D', 'S', 2021, 5, 1));**

**std::cout << "Passengers who will fly away at 2021.6.24:\n\n";**

**l.printAllWhere(isSuitableData);**

**l.del(Ivanov);**

**l.del(Petrov);**

**std::cout << "Other:\n\n";**

**l.printAll();**

**std::cout << "Last passenger: \n\n";**

**l.popBack().print();**

**}**

**Результат работы программы**

Passengers who will fly away at 2021.6.24:

Ivanov R.V.:

Destination: Moscow

Flight number: 324

Date: 6.24.2021

Petrov B.N.:

Destination: Paris

Flight number: 324

Date: 6.24.2021

Other:

Sidorova V.E.:

Destination: Kazan

Flight number: 144

Date: 7.14.2021

Elmov I.B.:

Destination: Antalya

Flight number: 574

Date: 8.24.2021

Chernov D.S.:

Destination: Vladivostok

Flight number: 574

Date: 5.1.2021

Last passenger:

Chernov D.S.:

Destination: Vladivostok

Flight number: 574

Date: 5.1.2021

**Выводы**

1. Я изучил механизм наследования классов в языке C++ и научился применять его в программировании.
2. Я получил навыки в разработке базовых абстрактных классов, а также в построении производных от них классов.
3. Я изучил механизм создания шаблонов классов и функций в языке C++, а также научился применять их при разработке программ.